December 2010

Design Patterns in User Interface Design

Eric Magnuson
Worcester Polytechnic Institute

Design Patterns in User Interface Design

A Major Qualifying Project

Submitted to the faculty

Of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

_____

Eric Magnuson, Class of 2010

December 16, 2010

_____	_____
Matthew Ward	Jeffrey LeBlanc
Project Advisor	Project Advisor

Abstract

The concept of Design Patterns has been around for over 30 years, originating in the architectural community. This paper explores the application of Design Patterns as it relates to User Interface Design. Current pattern collections and thinking are researched and evaluated to determine their overall effectiveness, as well as their practicality. A prototype application is developed using User Interface Design Patterns in order to demonstrate the utility of design patterns. Design patterns are found to be a useful development tool, however the lack of definition, simplicity of many patterns, and scattered pattern collections prevent them from realizing their potential in the User Interface community.

Table of Contents

Introduction

Since the late 1990's *design patterns* have been discussed in the user interface development community as a potential means for disseminating expert knowledge. The idea of using patterns for communicating domain knowledge originated with Christopher Alexander, a now-famous architect who hoped to improve the lives of the inhabitants of his architectural designs. The concept translated nicely to software design, where pattern usage was met with huge success. The discussions in the user interface community have centered on capturing this success, and extending it to designing and implementing more successful user interface designs.

Conferences held for the user interface community have been a hotspot for discussions of patterns. The first appearance of patterns at a conference was in 1997 at the CHI conference, sponsored by ACM SIGCHI. Since then there have been numerous published articles and books furthering the discussion, refining the definition of what a user interface pattern consists of, and introducing new patterns to the community. In 2010, patterns were given center stage with the PEICS conference, which focused explicitly on issues surrounding designing and engineering user interfaces using design patterns. Amongst the things discussed have been how to define and structure patterns, and how to take the patterns and implement them in an automated fashion.

A special focus is given to mobile user interfaces in this paper, as mobile devices continue to become more and more prevalent. Mobile user interfaces present unique challenges that are not present on desktop systems, due to the wide variety of input methods, small screens, and susceptibility to task interruption. They also present unique

opportunities, many of which stem from the same variety of user input methods such as touch screens, accelerometers, and GPS capabilities. Since designing mobile interfaces is a relatively new practice, and the field is under constant flux from the rapid advancements in technology, it is especially important to find an effective and efficient way to disseminate expert knowledge to the community.

The existing research on patterns in the user interface community is examined in this paper. Patterns have been around in the user interface community for over a decade, and yet their use in practice has been far more limited than many thought they would be. This paper aims to better understand the current discussions regarding design patterns, and to evaluate them to identify why they have not been utilized to their full potential. Additionally, the use of design patterns in practice is demonstrated by using patterns to design a user interface. The interface is then implemented in a prototype application, demonstrating the full process involved in developing user interfaces with design patterns as a core development technique.

Literature Review

Pattern Beginnings

Design Patterns were first introduced as a defined concept by Christopher Alexander in his two books *A Pattern Language* in 1977, and *The Timeless Way of Building* in 1979 (Seffah 2010). Alexander is an architect who envisioned a way to capture all of the best aspects of architectural design in an easy-to-understand collection of what he termed "Patterns." Doing so enables engineers, architects, and even the laymen who would be

using the buildings to communicate design ideas easily, and understand the problems facing each design.

Ultimately, Alexander wanted his pattern library to be used to help improve the quality of life for the people who would be living in or using the buildings. He hoped to capture what he refers to as *the quality without a name*, which he defined as follows: "there is a central quality which is the root criterion of life and spirit in a man, a town, a building, a wilderness. This quality is objective and precise, but it cannot be named" (Wania and Atwood 2009). Effectively, what this quality describes are those thoughtful designs that, whether obvious or subtle, make inhabiting a space a more pleasant, usable, or relaxing experience. What makes this emphasis so important is in where the focus of the design is placed. The central idea is to place the needs and desires of the inhabitants (or end-users) ahead of all other design decisions that might factor in.

The concept of inhabitant-centric architectural design in and of itself was neither novel, nor significant enough to make pattern usage such an important technique. Where patterns truly excel is in communicating the common problems that face designers, and providing potential solutions (Seffah 2010). Alexander states that every pattern has three essential parts: the context which describes a recurring set of situations in which the pattern can be used; the problem which refers to a set of forces which occur in the context; and the solution which refers to a design form or rule that can be applied to resolve the forces (Seffah 2010). By including these components, the pattern is guaranteed to address the core issues that a designer is looking to solve, and gives them a potential method for solving the problems. Another advantage to this is that the people who will eventually

inhabit the area can, at least partially, understand the components of the pattern so that they can take part in the design process, communicating their needs and desires to the designer in a more useful way.

Alexander's explanation and use of patterns, while innovative for the architecture community, was largely irrelevant to the software design community until the seminal book *Design Patterns* by Gamma, Helm, Johnson, and Vlissides (often referred to as the *Gang of Four* or GoF) was released in 1994 (Wania and Atwood 2009). This book took the principles of patterns and applied them in an entirely new scope, using them to describe common problems facing software designers, and providing methods for them to solve the problems. The solutions presented by the GoF were not new; in fact the whole idea was to capture the knowledge that most experts already had. What made this book so important was the fact that the knowledge these experts possessed was captured and could now be easily distributed to those less knowledgeable in a way that they could understand and readily use (Gamma, et al. 1995).

The patterns demonstrated in *Design Patterns* focused around two keys attributes; they had to be reusable and they had to be flexible. These keys were carefully and deliberately chosen, as the GoF also state that "Experienced object-oriented designers will tell you that a reusable and flexible design is difficult if not impossible to get 'right' the first time" (Gamma, et al. 1995, 1). This, again, demonstrates one of the driving forces behind the usefulness of patterns: communicating expert knowledge to others. By directly addressing some of the more difficult tasks to get done right from scratch, the pattern library greatly increases its utility to developers. Additionally, much like a good cook might use a recipe

book for inspiration, designers can apply the patterns to a variety of situations, even if the pattern may not originally have been intended to address that particular situation. This is made possible by the somewhat generic nature of the pattern itself, describing problems, solutions, and the forces acting on each, rather than the absolutes of a specific circumstance.

There are four essential elements to the GoF's structure for a pattern: pattern name; problem description; solution; and consequences. The pattern name must be descriptive and simple. This allows the pattern to be quickly and usefully referred to, giving a strong vocabulary to be used in design discussions and education. The problem description must detail when the pattern should be applied, and it should include any possible conditions that the pattern would depend on. The solution should abstractly describe the elements of the design, describing the relationships, responsibilities, and collaborations of each of these elements. Again, no particular concrete implementation is listed, allowing the solution to be interpreted to apply to a wide variety of similar situations. The consequences describe trade-offs that result from applying the pattern, including the impact on overall design flexibility, extensibility, or portability. (Gamma, et al. 1995)

The pattern library's utility is also enhanced by a more formal pattern definition and organization. Each pattern is described using the same 13 sections, which combine to address the four essential elements of a pattern, as well as details such as examples in use and sample code[1]:

- Pattern Name and Classification
- Intent
- Also Known As
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample code
- Known Uses
- Related Patterns

[1] See Appendix A for additional descriptions of each section

"The template lends a uniform structure to the information, making design patterns easier to learn, compare, and use" (Gamma, et al. 1995, 6). They also present a classification method for the patterns, categorizing and organizing them. The patterns are classified by purpose and by scope, creating a grid-like catalog that contains all of the patterns. "The classification helps you learn the patterns in the catalog faster, and it can direct efforts to find new patterns as well" (Gamma, et al. 1995, 10

Patterns in User Interface Design

Given the proven utility of patterns in both architectural and software design, it is no surprise that there has been a significant amount of interest in applying patterns in user interface design. One of the early demonstrations of this gaining widespread interest is the workshop "Putting It All Together: Pattern Languages for Interaction Design" given by Thomas Erickson at the CHI conference in 1997 (Seffah 2010). Since then, a number of books and other publications have detailed various aspects of design patterns as they relate to user interface design. Many of the discussions prior to 2001 focused on defining what a user interface pattern was, and what roles it had. Ahmed Seffah provides the following definitions in his paper "The Evolution of Design Patterns in HCI"

From the most generic to more HCI domain dependent, a HCI pattern is:

- *Form, template, or model or, more abstractly, a set of rules which can be used to make or to generate things or parts of a thing;*
- *A general repeatable interaction technique to a commonly occurring user problem;*
- *"An invariant solution to address a recurrent design problem within a specific context" (Dix, 1998);*
- *A general repeatable solution to a commonly-occurring usability problem in interface design or interaction design;*

- *A solution to a usability problem that occurs in different contexts of use;*
- *"A successful HCI design solution among HCI professionals that provides best practices for HCI design to anyone involved in the design, development, evaluation, or use of interactive systems" (Borchers, 2001).*

Each of these definitions has significant similarities to the definitions Alexander and the GoF used to describe their patterns, despite describing how to utilize patterns in a different domain. This demonstrates how pervasive and universal the pattern concept is. Currently two other concepts for disseminating expert knowledge, guidelines and claims, are more predominant in the domain of user interface design. Guidelines are frequently used to promote consistency in the look-and-feel of different applications on the same platform. Claims are used to provide design advice based on a specific usage context, including design rationale and context-of-use scenarios (Seffah 2010). These methods are limited though, as patterns provide greater ability to disseminate a working knowledge of reusable design solutions, as demonstrated by the architectural and software design patterns. Additionally, patterns have the advantage of combining the theoretical design rationales with concrete examples and implementations, giving the reader all of the information required for applying the design.

User interface design is also unique in the sense that it pulls knowledge from a number of very different disciplines. Design involves the user interface design expert, the system designers, and application domain experts, at the least. The knowledge from these disciplines is not always easily communicated to the experts from the other disciplines. Therefore, there needs to be some way for these experts to effectively communicate their knowledge to each other. Pattern languages provide this communication, and can act as a

Lingua Franca for design teams (Borchers 2001). Increasing the effectiveness of communication in turn increases the utility of the knowledge conveyed. Keeping with the principle of user-focused design, this allows designers to better incorporate domain-specific knowledge into their designs, and make them work more effectively with the system they are being designed for, providing a greatly improved user experience. It also greatly reduces the efforts and stresses of generating a design in the team, since the whole system is better understood by the individuals on the team.

Jan Borchers provides a proof of concept for using patterns in each aspect of a system design, and as the *Lingua Franca* for his design team, in his paper "A Pattern Approach to Interaction Design" where the team is designing interactive music exhibits for a museum. This project involved knowledge of music, exhibit design, interaction design, and software design. Experts in each domain were required to express their knowledge in the form of patterns in an agreed upon format. The results from this were very promising. They found that by expressing the knowledge in such a way, experts were forced to consider their knowledge more carefully, and were able to provide a more fully developed description of their knowledge. This in turn provided non-experts a better understanding of the knowledge through the pattern. Additionally, they found the patterns to be highly reusable. Within the same project, they were able to slightly modify and reuse patterns they had implemented. Additionally, when they created similar systems, they were again able to reuse and refine the patterns to suit their needs, and provide them with needed domain knowledge. (Borchers 2001)

Borchers also demonstrates how design patterns fit into the 11 activities suggested by Nielsen's Usability Engineering Life Cycle model:

- Know the User
- Competitive Analysis
- Setting Usability Goals
- Parallel Design
- Participatory Design
- Coordinated Design of the Total Interface
- Apply Guidelines and Heuristic Analysis
- Prototyping
- Empirical Testing
- Iterative Design
- Collect Feedback from Field Use

A few of these stood out in particular. Patterns achieve "Know the User" by providing an effective way for domain experts to communicate their knowledge in a consistent and accessible manner. "Setting Usability Goals" is achieved by using the goals as forces when describing the patterns, ensuring that they are always being considered. "Parallel Design" is achieved by utilizing the patterns to simultaneously create alternate, but consistent, designs. By allowing users to see, understand, and create patterns, designers can easily achieve "Participatory Design." Each activity can be described or achieved in terms of design patterns, demonstrating their utility in yet another way (Borchers 2001).

Jenifer Tidwell provides one of the more complete collections of user interface patterns in her book "Designing Interfaces." She defines patterns as a description of best practices within a given design domain. They don't provide off-the-shelf components, nor are they just a set of simple rules. She provides her patterns not as part of a recipe for creating user interfaces, but rather as a way to learn design techniques, see examples, adopt terminology, and hopefully find inspiration for better interface design. She also provides the patterns in an organized fashion, dividing the patterns into various categories that best describe their use. In doing so, she makes it much easier to find a pattern which solves a particular problem. (Tidwell 2006)

Tidwell's patterns follow a specific format, much like the patterns in Alexander's "A Pattern Language." Each pattern has a descriptive but concise name for easy reference. The first item in each pattern is the "What" which briefly describes what the pattern does. "Use When" describes the forces that should be considered when deciding whether to use the pattern or not. "Why" explains what makes the pattern effective, helping the reader to understand the design principles involved, as well as gain a better understanding of when the pattern is most useful. "How" helps to explain some of the implementation level details, addressing common pitfalls and issues that a designer might experience in trying to use the pattern. (Tidwell 2006)

One of the more interesting and unique uses of patterns by Tidwell is in her description of human behavior in terms of patterns. She identifies 12 common behaviors which are pertinent to user interface design. Unlike the other patterns in the book, these are not descriptions of a problem, context, and solution, but rather a description of how people

tend to approach and use software. She advises that "an interface that supports these patterns well will help users achieve their goals far more effectively than interfaces that don't support them" (Tidwell 2006, 10). While these patterns don't provide specific interfaces, nor do they solve specific problems, knowing them helps the designer to understand the user a little better. Defining these behaviors as patterns gives them a common name, and helps identify them and implement them in a design. (Tidwell 2006)

User Interfaces in Mobile Devices

Mobile devices present many unique challenges for a user interface designer that are either uncommon or absent from traditional desktop computing. While many of the overarching design principles remain the same, the mobile device interfaces demands some special consideration, especially given the explosion of mobile software due to devices such as the iPhone (3 billion downloads)[2], Android(1 billion downloads)[3], and Symbian (3 million downloads per day)[4] phones. In order to make the best interfaces, the designers must acknowledge and learn the restraints and advantages involved in mobile computing, and incorporate this information into the design accordingly.

One of the problems designers face in the mobile realm is the wide variance of capabilities from device to device and platform to platform. One phone may have a touch screen and no buttons, another may use a full QWERTY keyboard, and yet another may only have a number pad and a few arrow keys. These variances are equally different when

[2] http://www.apple.com/pr/library/2010/01/05appstore.html

[3] http://www.informationweek.com/news/hardware/handheld/showArticle.jhtml?articleID=225800262

[4] http://conversations.nokia.com/2010/11/18/ovi-store-3-million-downloads-a-day/

it comes to computing power, screen capabilities, and more. To accommodate these differences likely requires building multiple designs for each platform the interface needs to run on.

Barbara Ballard recommends building a device capability hierarchy in order to reduce the number of designs that need to be produced. This hierarchy organizes devices so that the root node of the hierarchy is the highest-impacting capability, usually screen size. The hierarchy progresses towards the least-impacting capability. The leaf nodes of this hierarchy contain all of the design patterns that are still usable for the given tree traversal. In comparing the patterns available to the various devices that are being designed for, the designer can identify interface patterns that are common to all of the devices, and build a design accordingly. The biggest limitation associated with this method is that there is no easy or automated way to match devices to this hierarchy, which means the designer must manually identify all of the capabilities and limitations of a device and associate them with the hierarchy accordingly. (Ballard 2007)

One of the significant differences in mobile design is the idea that mobile applications are get-in-get-out type applications. Limitations on the mobile platform, and discomfort that comes with prolonged use, prevent mobile devices from being extended use devices. This dictates that mobile applications be smaller, and more task specific than desktop applications. Given this, it's crucial to mobile applications that the user is able to get to what they need to do as quickly as possible. According to Ballard, "Users tend to want to get their content, including download and purchase if relevant, within 20 seconds; some data suggests that the impatience limit is actually below 10 seconds" (Ballard 2007). It is

reasonable to assume that as devices become more capable, the demand for even faster launch times will increase as well. This principle goes well beyond the launch process itself, and should be present in all of mobile design. If it's possible to reasonably avoid a step to complete a task, then it should be avoided. Common tasks should be quickly accessible immediately upon opening the application. Computing power is not present on most mobile devices, so offloading heavy computing tasks to external servers is an ideal solution when possible. These are just a few examples of how to speed up mobile applications to meet the user's needs.

Another characteristic of mobile devices is that they are inherently highly interruptible. Mobile devices are typically used out in the world, not in a controlled office environment as desktops would be. This means it is far more likely that some environmental factor will distract from the task at hand. Additionally, users may be using the mobile device while performing other tasks such as walking or holding a conversation ,which means the application will not receive the user's full attention. Mobile applications need to accommodate this in their designs. One way to address this is to design interface elements so that they require minimal attention, either by making them larger, or simplifying the choices a user needs to make (Kane, Wobbrock and Smith 2008). Another technique is to make sure to save context where appropriate, allowing the user easy reentry into the task they were distracted from (Tidwell 2006). Reentrance is especially important when you consider the user's need to access and complete tasks as quickly as possible.

User input, text entry in particular, is another design issue that is significantly different on mobile devices. As previously mentioned, mobile devices have a wide variety of input

methods. However, one commonality between them all is that they are much smaller than the traditional keyboard, and therefore much slower and harder to use. It is important to understand this, and to incorporate this understanding into the interaction design to make it easier for the users to enter text and data into the application. Erik Nilsson presents a number of design patterns to address this. "Autocomplete" is used to reduce the need to enter text by predicting what the user is trying to type, and suggesting it. "Alternative Input Mechanisms" and "Specialized Input Mechanisms" take advantage of the unique input methods and direct manipulation capabilities of mobile devices to present elements such as check boxes, spinners, and manipulable objects for the user to interact with in place of text input (Nilsson 2009).

From Theory to Practice

Despite the many apparent advantages of designing with patterns, the idea has not truly caught on in practice the way many expected it would. They remain great classroom tools, but designers have not been able to utilize them to their full potential. One reason for this is that there has yet to be any standardization in the way the patterns are formatted. This makes it very difficult to build a dynamic collection of patterns. In a similar vein, there is no agreed upon organization for the patterns. Organization is key to a designer's ability to search a pattern collection and find a pattern that addresses the design problem. Without strong organization, and well-defined relationships between patterns, a pattern library is significantly less useful to designers. In part as a result of these, there are very few tools that support designers in discovering and implementing patterns. There is no way to computationally process patterns because of the lack of a standard presentation. This

makes it more difficult to develop for reusability, since there is no way to dynamically generate code. (Breiner, et al. 2010)

These obstacles have not completely thwarted the implementation of pattern libraries however, as is detailed by Matt Leacock, Erin Malone, and Chanel Wheeler, developers at Yahoo! They describe their process for implementing a user interface pattern library that would be accessible and used by user interface designers within the company. They designed and built a repository that would be scalable, easy to use, and encourage collaboration amongst its users. They also defined a format that each pattern needed to adhere to, giving standardization within the scope of their library. The repository also defined categorization for the patterns, so that designers could easily discover new patterns that would be useful to them. In addition to this categorization, patterns were rated by designers on a scale that factored in how important it was to the company that the pattern be adhered to. They also considered rating scales based on the strength of evidence that the pattern was useful to the end users, as well as the quality and clarity of the pattern definition, but decided against using them in their repository. One thing to note about their pattern library is that it does not include any code or implementation of the patterns, but instead acts as a reference to designers as to how to solve the various problems they may commonly face. Some of the patterns developed using this repository can be found at http://developer.yahoo.com/ypatterns/ (Leacock, Malone and Wheeler 2005).

As is always the case, no user interface design is complete without thorough testing and evaluation. Simply using patterns does not guarantee a successful design. When considering the relatively new field of mobile application design, testing becomes even

more important to help identify and eliminate unforeseen problems. To this end, a new set of heuristics for mobile computing was investigated by Enrico Bertini, Silvia Gabrielli, and Stephen Kimani. They evaluated Nielsen's traditional heuristics in order to determine which ones were applicable, needed modification, or were irrelevant in terms of mobile user interfaces. They agreed upon the following mobile user interface heuristics: [5]

- Visibility of system status and losability/findability of the mobile device
- Match between system and the real world
- Consistency and mapping
- Good ergonomics and minimalist design
- Ease of input, screen readability and glancability
- Flexibility, efficiency of use and personalization
- Aesthetic, privacy and social conventions
- Realistic error management

They found that by tailoring heuristics specifically towards mobile computing, they were able to identify more design flaws than using Nielsen's original heuristic evaluation methods (Bertini, Gabrielli and Kimani 2006). This is particularly useful in identifying problems early in the design phase, before their impacts become too significant.

User testing must also be adjusted to account for differences in mobile computing. Traditional user testing occurs in a controlled lab environment, where various effects can be recorded and measured in exact quantities. Mobile computing defies this logic however, because of the inherently unpredictable environment that the application will be used in.

[5] See Appendix B for a full description of their mobile heuristics

"Running usability studies in public spaces reveals issues that cannot be found using lab studies, as studies in public spaces are invariably subjected to environmental effects" (Kane, Wobbrock and Smith 2008). In order for a user test to be truly complete, it must account for not only the user, but the likely environments and circumstances the user might use the application in.

Designing the Interface

Selecting an Application

The first step in any design is to identify the goals that the design should accomplish, and to identify any constraints that may be imposed on the design from external sources. In this case, the primary goal of this design is to demonstrate the process of building a user interface using design patterns. The most significant constraint involved is time. Given the lack of time, it is not possible to both build a working prototype of the application and do user testing to demonstrate its effectiveness. Instead, the application must be simple and familiar, so that its effectiveness is self-evident through previous experiences with similar applications. The time constraint also means that the design will not be fully realized in the prototype; however, the prototype will still have to demonstrate the utility of design patterns.

Another goal for this interface was for it to be applied to a mobile device. Mobile applications are exploding onto the market right now, and this provides a good way to demonstrate design patterns in a very active market. Specifically, the application was designed for the Nokia N900.

Using these factors as the guide, the application I decided to design is a simple instant messaging client. This is a very familiar paradigm, as many people have used some form of instant messaging client, and this familiarity will help to demonstrate the effectiveness of the design. In addition, the design is very simple, allowing for a working prototype to be built within the given time frame. Despite its simplicity, there is still some room to introduce unique features to the interface should there be an appropriate pattern that can be applied.

Understanding the Design Issues

Before we can explore the patterns, there needs to be some idea of which patterns might be beneficial, and which ones would be of no use for the given application. We must identify and understand the design "problems" – or user tasks – that need to be solved. The instant messaging application has a number of tasks that are central to its operation, and for which design patterns might help us to build a user interface.

The primary purpose of the application is to be able to send and receive messages with another contact. The messages should be presented to the user legibly, and in chronological order so that the overall conversation can be understood. The user should be able to have the ability to review at least some messages from the history of the conversation, although it is preferable that the full transcript be available. It should be obvious to the user how to go about sending a message.

Another key component of the application is that the user must be able to hold several independent conversations simultaneously. Each conversation should have its own independent space that will not interfere with the other conversations. It should also be

obvious to the user which contact the conversation is being held with. Switching between conversations should be trivial, as this is likely to be a frequent action while using the application.

Similarly to this, a user should be able to view a full list of their contacts. This list should provide some level of user sorting or grouping so that contacts are easier to find. The user should be able to use this list to initiate a conversation with one of their contacts, or activate an existing conversation. Depending on the style of messaging network, this list should also be user manageable, allowing the user to add and remove contacts at will. In some closed systems, it may be reasonable for the contact list to be delivered by the system (i.e. a closed work system), but the contact list organization should still have some level of user control, even if as simple as a "favorites" group.

By understanding these core problems that our design needs to solve, we can gain a better perspective on which patterns are going to be applicable to our design.

Selecting the Patterns

In choosing patterns to use for this application, I didn't work from any one library or collection. Instead, I pulled from all the sources that I had read to identify patterns that could be useful for the instant messaging application. One of the drawbacks to this is that the patterns do not all adhere to the same format, or solve the same type of problems. Some of the patterns address layouts, others address interaction methods, and still others address application behaviors. However, all of them have a problem that they describe, and provide a solution that best solves the problem.

Figure 1 - The Nokia N900

Before looking through the patterns, I evaluated the Nokia N900 to identify the attributes it has which would affect my pattern selection. The N900 is a touch screen mobile phone. It can be operated using finger touch, or by using a stylus. It also has a slide out hardware keyboard for text entry. The screen measures approximately 2 inches by 3 inches. It has wireless lan (WLAN) and cellular network connectivity. The most important factors for my interface design are the slide out keyboard and small, touch capable screen.

Finger Friendly Controls

This pattern comes from Erik Nilsson's paper "Design Patterns for User Interface for Mobile Applications" (Nilsson 2009). This pattern is simple, but important for mobile devices. Any object on screen that the user needs to manipulate via direct touch with their fingers, needs to be sized in such a way that it can be easily manipulated with the fingers. There are two major factors at play with finger-based manipulation: the finger is a broad pointing device, lacking the fine point that a stylus benefits from; the finger, and the hand in general, provides more obstruction to the screen than a stylus. This means objects must be made a bit larger, to accommodate for the finger's inherent inaccuracies. Nilsson describes

the specific case of bringing up a menu as an example of how the pattern can be implemented. He suggests: “an alternative to standard menus or buttons that are always visible is to provide menu choices in a small popup panel at the bottom of the screen” (Nilsson 2009).

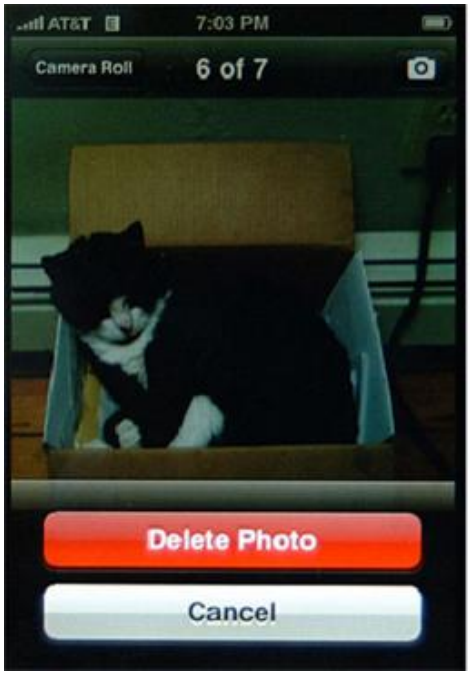

Figure 2 - Finger Friendly Menu example from Nilsson's paper

This pattern is an easy choice, given that I am designing for a mobile touch screen device. Implementing this pattern will require that any on screen object that the user will need to interact with be at least the physical size of a fingertip, which is approximately

14mm by 18mm.[6] Objects that are not going to be interacted with, or will be interacted with exclusively using the keyboard do not need to meet this minimum size requirement.

Two-Panel Selector

The two-panel selector is a content-organizing pattern designed to provide a list of selectable items alongside the content for those items. It is achieved by dividing the window content into two distinct panels. In one panel, either above or to the left of the second panel for left-to-right reading cultures, we provide a list of the items. In the other panel, we provide content that changes according to which item is selected. (Tidwell 2006)

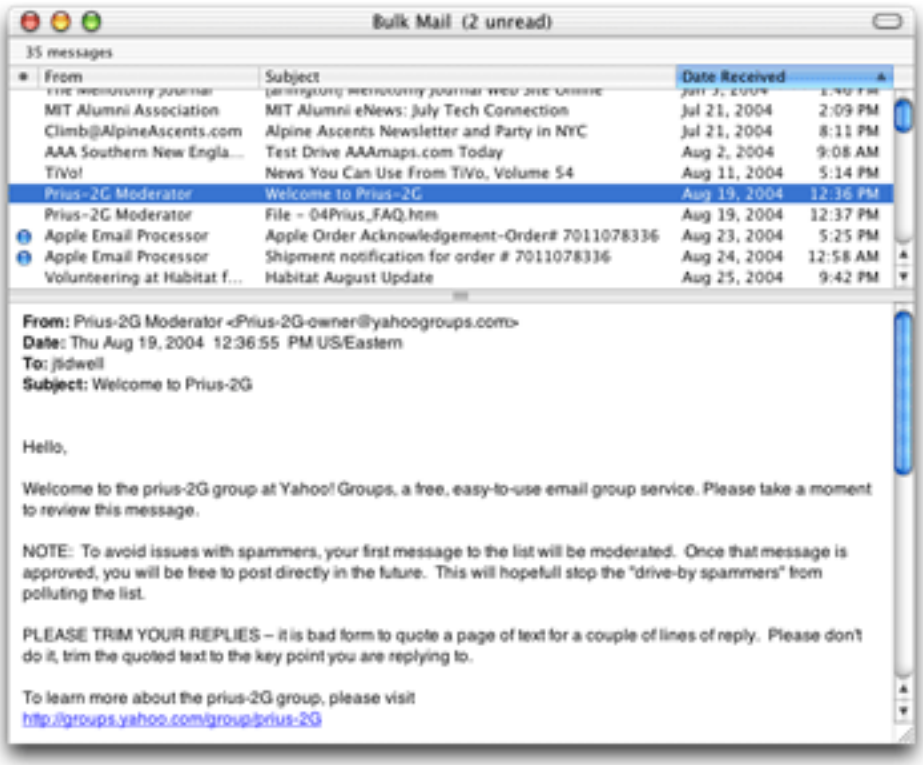

Figure 3 - Two Panel Selector example from "Designing Interfaces"

[6] Approximated from Table 6 in the paper "Empirical Evaluation for Finger Input Properties In Multi-touch Interaction" by Feng Wang and Xiangshi Ren, published at the CHI2009 conference.

This pattern would be a useful way to present the contact list and conversation windows, given the natural one-to-one nature between selecting a contact and displaying a conversation with that contact. It also makes a very good use of the screen space on a mobile device. Since the screen is laid out with a wide screen when in landscape mode, there is enough room for both panels to exist side by side. This pattern reduces the cognitive load involved in switching between different contexts, in this case the conversations. This is particularly important since the nature of instant messaging encourages frequent context switches. Two-Panel Selector is also a very familiar paradigm, learned from programs such as email clients or file system navigation such as Windows Explorer. It provides an added potential benefit in our specific context in that the contact list can reflect changes that occur in the conversations, drawing attention to background conversations which have unread incoming messages for example.

Hub and Spoke

Hub and Spoke is a navigational pattern found frequently on mobile devices. It provides for a method of switching between several discrete tasks within the application by providing one way in to the task, and one way out to the task selection. If you were to map this out in a chart, it would result in one central object with several other objects surrounding it, with direct lines back to the central object, the result of which resembles the appearance of a spoked wheel. It frequently used as a method for launching applications on a mobile device, with a central home screen listing the applications, and each application returning to the home screen when it is exited. (Tidwell 2006)

Figure 4 - Hub and Spoke example using Nokia Series 60 phone from "Designing Interfaces"

In the case of the instant messaging application, the most frequent context switching occurs between conversations. The hub and spoke pattern could be applied to the conversation windows to provide a quick method for switching between them. There could be a way to enter into the "hub" mode, displaying all of the open conversations for the user to select from. Once selected, that conversation window would then be brought to the foreground. This is a slight modification on the pattern as Tidwell describes it, since it is not being used to force any task completion. But it provides a clean method for switching between conversation windows, and it eliminates the drawbacks of sequentially switching conversations by providing a space for the user to select which conversation they want to view.

Card Stack

The Card Stack design pattern places sections of content onto discrete panels, with only one visible panel at a time. The interface provides tabs or other similar methods for the user to be able to switch between the various panels. It is useful for situations where the

content is easily broken into discrete sections, and only one section is needed at a time. It is also a very familiar paradigm, seen frequently in Microsoft settings dialogs, or modern day tabbed browsers. (Tidwell 2006)

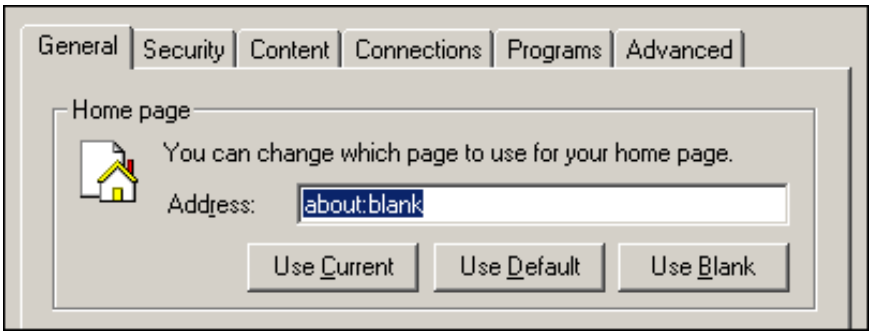

Figure 5 - Card Stack example from "Designing Interfaces"

This pattern would provide another means of switching between the open conversations in the instant messaging application. This could be done by providing a tab for each open conversation, allowing the user to select a conversation quickly and easily. Another advantage to this is that the tabs could be used to indicate other things, such as a new unread message. However, given the screen real estate restrictions, it is likely that tab space would run out quickly. In this case, providing a dropdown menu for the user to select an active conversation from would likely be a more space-friendly method of providing the Card Stack.

Application State Management

This pattern, described by Barbara Ballard, is an example of an application's behavior contributing to the interaction design. The pattern describes the different ways the application should be managing occurrences such as exiting the application, or switching tasks mid-application. If the application is exited, it should remember the state it was in so

that it can return the user to the most logical point. This may be the exact screen it was on when the application was exited, or it may be a central screen, depending on the application and the state it was saved in. Additionally, user input should always be saved, and only discarded after the task is completed or explicitly cancelled by the user. (Ballard 2007)

In mobile devices in particular, tasks are frequently interrupted. These interruptions can come from the device itself (i.e. receiving a phone call), or by an external source (i.e. someone knocking on the door). Instant messaging applications also encourage frequent task interruptions, since background conversations are still receiving incoming messages that may demand the user's attention. Given these properties, it is important for the application to always return the user to their previous state. In the case of exiting the application, this means that the user should be brought back to the same conversation they had open, and with the same open conversations in the background. The conversations should all maintain the transcript of the previous messages, and not start from a clean slate. Internally, if a user switches between conversations, any text they had started to type out should be saved, and be ready for them to pick up where they left off when they return to that conversation. As with exiting the application, each conversation should always maintain its transcript.

These patterns highlight just a few of the many patterns I identified for use with the instant messaging application. Ultimately, I identified more than 20 patterns that could potentially be integrated into the interface design. Most of these are able to be used

simultaneously, while a few of them, such as the Card Stack and Hub and Spoke patterns, may solve the same problem, and thus would conflict with each other.

Applying the Patterns

Using the set of potential patterns identified, it is now possible to apply them to a realized interface design. In a real development environment, the patterns might be combined in different ways in order to form multiple prototype designs, which would then be evaluated and narrowed down until a final design was reached. In this case, the patterns will be used to build a single design in order to demonstrate the use of design patterns in producing a user interface design.

The core elements of the design have already been identified – a contact list and conversation windows. As such these elements will take the primary focus of the design. The *Two Panel Selector* pattern provides the best way to achieve this. The contact list is placed on the left portion of the screen, with the currently active conversation placed to the right. Since the conversation demands more attention, and naturally needs more space, the panels are split so that the contact list takes up one third of the available horizontal space, and the conversation consumes two thirds. Both the contact list and the conversation panel will have a limited amount of vertical space. While not necessary all the time, they each will need to be able to use the *List-Based Layout* pattern (Ballard 2007) in order to provide scrollable views of the contacts and messages.

Within the conversation panel, there needs to be an obvious point where the user will enter their messages. In combination with this, there should be a send button that adheres to the *Prominent "Done" Button* pattern, placed at the end of the visual flow, large, and well

labeled so as to make it obvious. (Tidwell 2006) Since the user will be typing out messages they wish to send, the return key on the keyboard should also be used to send messages, so the user does not have to switch between the keyboard and screen controls unnecessarily. When entering text into the message, the application will use the *Autocomplete* pattern (Nilsson 2009) to provide intelligent suggestions as to what word it thinks the user is typing, in an effort to reduce keystrokes on a mobile keyboard which users may find more difficult to use. Additionally, using the *Application State Management* pattern, if the user has a message typed out and switches conversations, that message will still be there when they switch back to the original conversation, ready for them to send or edit.

The messages will be displayed in such a way that it is obvious whether the user sent the message, or the message was received from the contact. Each message will also be able to display floating window following the *Datatip* pattern (Tidwell 2006), providing extra information about the exact time the message was sent. The panel will use the principles of the *Titled Sections* pattern (Tidwell 2006) to prominently display the name of the contact at the top of the panel, so that the user can clearly and easily determine which contact the conversation is being held with. If any errors occur, such as a message being unable to be sent, error text will be displayed inline with the conversation, adhering to the *Same-Page Error Messages* pattern (Tidwell 2006). If the error involves a message failing to be sent, the application will provide an option for the user to try to resend the message.

The contact list will list all of the contacts in user-specified groups. Each group will be listed as a *Closable Panel* (Tidwell 2006) so that each can be expanded or collapsed according to the user's desires. This allows the full list of contacts to be displayed, or the

user can choose to give focus to one or more sections as they choose. The titles of the panels will be the group names, and next to each name will be a number indicating the number of contacts that are in the group. Since a mobile screen does not provide much space, the contact list may induce a significant amount of scrolling. To reduce this, using the *Dynamic Queries* pattern (Tidwell 2006), a search box will be provided where the user can type in a contact's name. As the user types, the list of contacts will change to show only those contacts matching the search query. To indicate to the user what they should do with this search box, and to save screen space, the *Input Prompt* pattern (Tidwell 2006) will be applied to place ghosted text in the search box reading "Filter Contacts…" which will disappear once the user starts typing in the box.

Following the *Two Paneled Selector* pattern, selecting a contact from the list will open a new conversation with that contact, unless one already exists, in which case it will activate that conversation. Additionally, the contact list uses a variation of *Smart Menu Items* (Tidwell 2006). Instead of dynamically changing menu items however, a dynamic group titled "Active Chats" which lists all the contacts that have open conversations, providing quicker access to the current conversations taking place.

To make the two panels flow better, the *Diagonal Balance* pattern (Tidwell 2006) will be applied. This pattern arranges interface elements in an asymmetric fashion, balancing it by placing weight in the upper left and lower right corners. In this case, the "Filter Contacts…" prompt will be placed in the upper left corner, with the contact list below it. The message entry box will balance this in the lower right corner, with the list of messages appearing above it.

Extra actions that aren't supplied directly in the contact list or conversation interface elements are provided by a drop down menu as described in the *Finger Friendly Menu Choices* pattern (Nilsson 2009). A single menu is visible in the system title bar, which when pressed drops down an overlay listing the actions. The most common actions are displayed immediately in this menu, such as closing the current conversation. Other actions, such as managing the contact list, are provided using the *Modal Panel* pattern (Tidwell 2006), where a separate panel is displayed providing the tools to complete the action, and dismissed when the action is completed or cancelled by the user.

When the user first launches the application, they will be greeted with the *Wizard* pattern (Tidwell 2006) that will guide them through the initial setup of the application. After this first launch, the application will always launch to the screen it was on when the user last left it. Realistically, there may be some overhead involved during the launch process, such as reconnecting to the network, which would prevent the user from interacting with the program right away. The *Launch Process* pattern (Ballard 2007) suggests that the user wants to get to their task as quickly as possible, without wasting time with things like splash screens. In order to balance this with the necessary overhead and the need for branding, an image of the last screen will be saved. This image will then be displayed when the application is launched again so that the user can adjust to the context, but greyed out to indicate it is not active yet. An overlay will be displayed with the application branding, and stating what it is doing that is preventing the user from interacting at that moment, in line with the *Inform the User About What is Happening* pattern (Nilsson 2009). As soon as the overhead tasks are complete, the interface will be activated.

The design also adds in a unique feature to the instant messaging application using a variation of the *Color Coded Sections* pattern (Tidwell 2006). This pattern uses color to identify which section of an application the user is in, when there are distinct sections that need a different appearance either because they serve different purposes or for stylistic reasons. To apply this pattern in the instant messaging application, each group is given a user-defined color. All of the contacts in the group then take on that color, so they can easily be identified with their group. Similarly to this, conversations held with that contact are colorized to indicate which group they are associated with. This provides a subtle but strong reinforcement so it is easier for the user to recognize the context of their conversation. This helps reduce errors, especially potentially embarrassing ones where a message intended for a contact from one group is sent to a contact from another (i.e. sending a personal message to a work contact).

Final Design

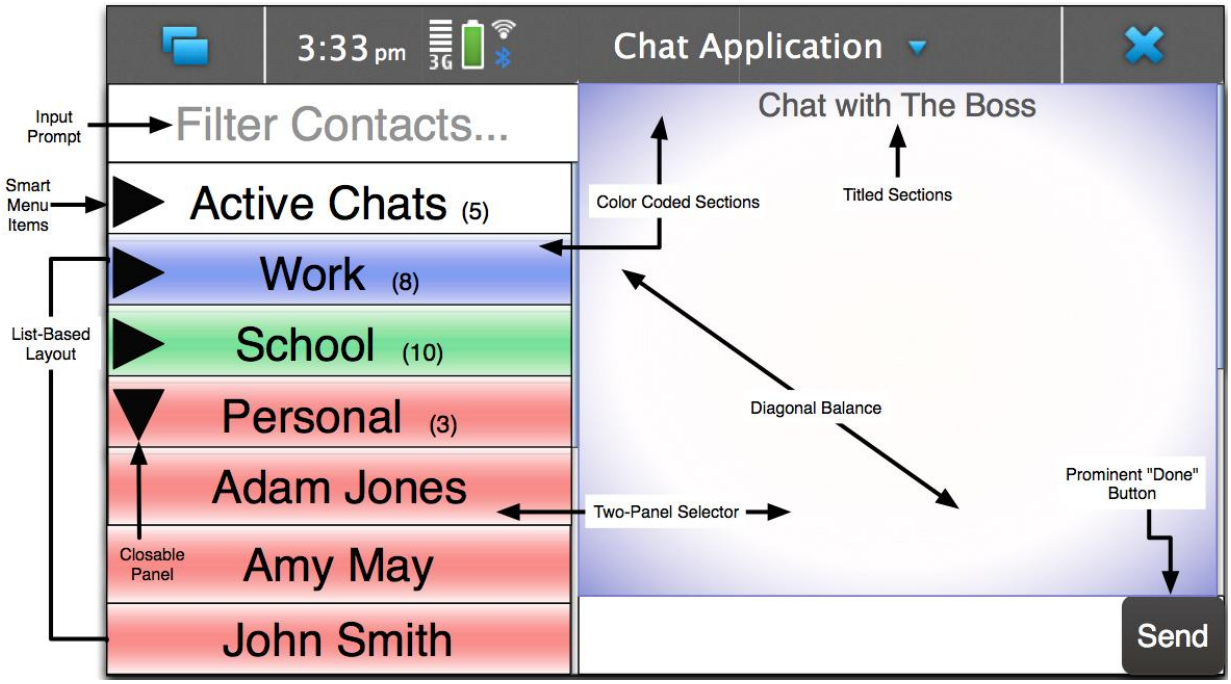

Figure 6 - Final Interface Design

Figure 6 shows a mockup of the final design for the interface design, as it would appear on an N900 device. The conversation panel is evident on the right hand side, titled to indicate the contact that the conversation is being held with. The background of the panel is colorized to indicate that the conversation is being held with a work contact. At the bottom of the window there is an obvious area for the user to enter their message, and a prominent "Send" button for them to send the message. The interface takes advantage of the N900's built in keyboard, so it doesn't need to bring up an on-screen keyboard overlay.

On the left hand side of the interface is the contact list. Collapsible groups show their associated colors in the background of the list item, and a small number to the right of the group indicates the number of contacts in the group. At the top of the contact list is the

"Active Chats" dynamic group, which contains all of the contacts associated with the currently open conversations. Above this is the "Filter Contacts…" search box, which allows the user to dynamically filter the list of contacts by typing their name.

In the upper right, the "Chat Application" menu contains access to the alternate actions for the application. The menu list contains actions for managing contacts, closing the current conversation, and setting other preferences. The prominent X to the right of this menu exits the application, as is the standard for the N900 interface.

Implementation

The Qt Framework

The Nokia N900, along with numerous other Nokia phones and other software, utilizes the Qt framework. The Qt framework, originally developed by Trolltech who was later acquired by Nokia, is an open source cross-platform application development framework.[7]

The framework's biggest strength lies in its cross-platform nature. Qt classes provide developers with all of the tools they need to implement applications of all types, and handle the platform dependencies internally. This enables them to develop one set of code and have it functioning on multiple platforms simultaneously, with no extra maintenance. Qt's classes are also very powerful, with classes for working with SQL database code, web programming using Webkit, scripting, GUI programming, and more. Since it is a framework, and not a library, Qt has the added advantage of being able to implement Qt classes in a

[7] Nokia also provides a special Commercial Developer License which includes support for developing with Qt and the ability to modify the source code without having to publish the modifications.

more ad hoc manner, tying in Qt classes with the native libraries. If there are platform specific libraries, Qt won't prevent developers from taking advantage of them.

Qt also provides extensive user interface development capabilities. A significant portion of the code involved in building and decorating user interface elements is handled by the Qt classes, allowing developers to utilize them with simple function calls. It also produces interface elements that are dynamically changed based on the platform the application is being built for. This is important in developing for multiple environments, as it lets the application take on the look and feel of the environment it is running in.

The user interface programming is enhanced by Qt Designer, which allows WYSIWYG editing of interface screens, referred to as "forms." Designer provides a number of built in options for elements to place on the screen, such as text browsers, lists, spacers, buttons, and containers. Beyond the built in items, the framework is highly extensible, allowing developers to implement their own custom interface elements. A plugin-based system also allows developers to implement and add in more programmable interface elements.

Nokia provides the Qt Creator IDE to tie all the tools of Qt into a single environment. Creator integrates Qt Designer to provide a user interface design tool. It also performs code completion, making coding faster and reducing errors. It integrates seamlessly with the Qt documentation, providing both context-sensitive help as well as full documentation and examples. Creator also manages the Qt project files and various build settings, making it even easier to produce a Qt application for multiple platform environments. Also included as part of the Nokia Qt SDK is the Qt Simulator. The Simulator emulates mobile devices running the Symbian or Maemo operating systems. It also is capable of emulating

environmental variables such as cellular signal, GPS coordinates, battery levels, and incoming events such as phone calls or SMS messages.

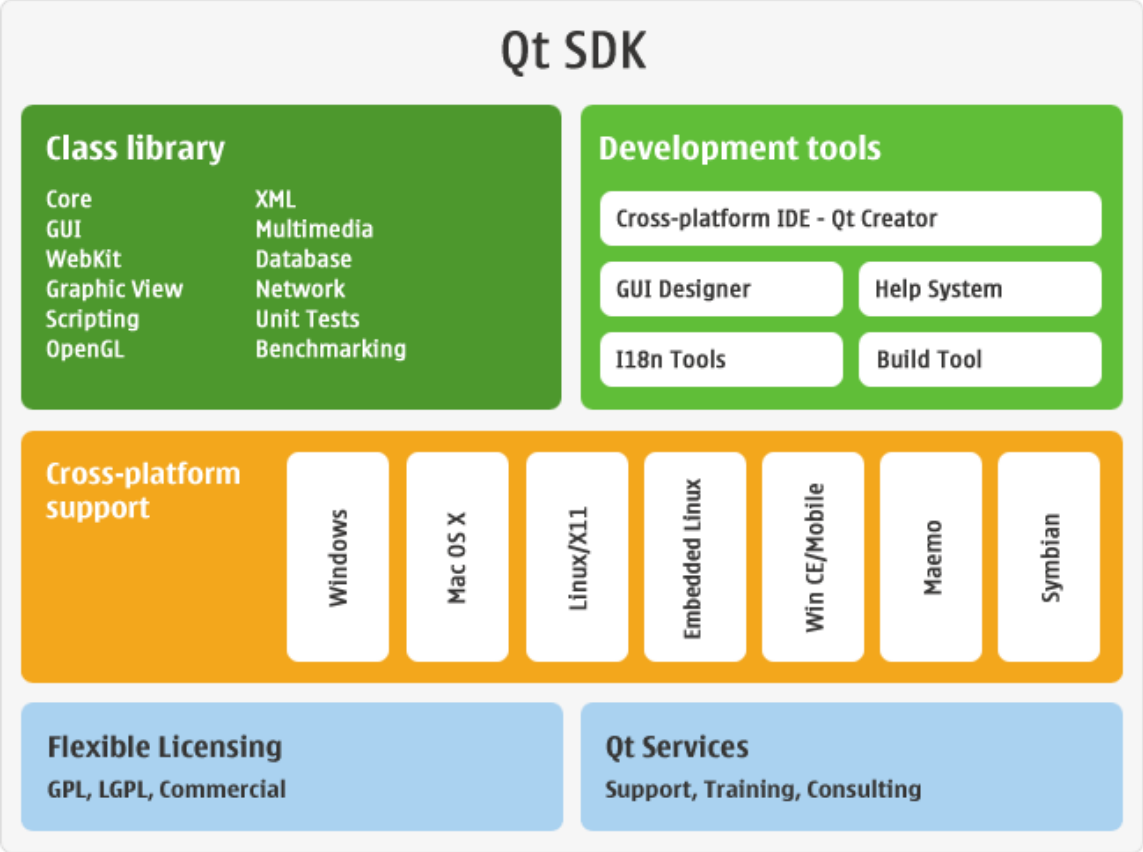

Figure 7 – A chart of what the Qt SDK provides

For this project I used the Nokia Qt SDK with Qt Creator 2.0.1 running in 64 bit mode on Mac OS 10.6. The Qt code base used was Qt 4.7.0. I developed for a Nokia N900 running the Maemo 5 operating system, with firmware version 20.2010.36-2.002 and Qt 4.7.0.

Implementing the Prototype

The primary goal in implementing a prototype application is to demonstrate the full process of using user interface design patterns to produce an application. While it is always preferable to abstract out the patterns into their own reusable objects, this is not always possible with user interface design patterns. One reason for this is that not all of the

patterns are programmatic in nature. Some patterns describe overall application behavior, or interface layouts, which are more effectively implemented in other ways. Other patterns are common enough that they are handled automatically by the interface components themselves. For example, Qt fully or partially implements many of the patterns I chose for my design, including:

- List-Based Layout
- Closable Panels
- Dropdown Chooser
- Modal Panels
- Datatips
- Finger Friendly Controls
- Autocomplete
- Application State Management

The first step to implementing the prototype is to produce a layout using the forms in Qt Creator. Some of the patterns will be implemented by completing this step alone.

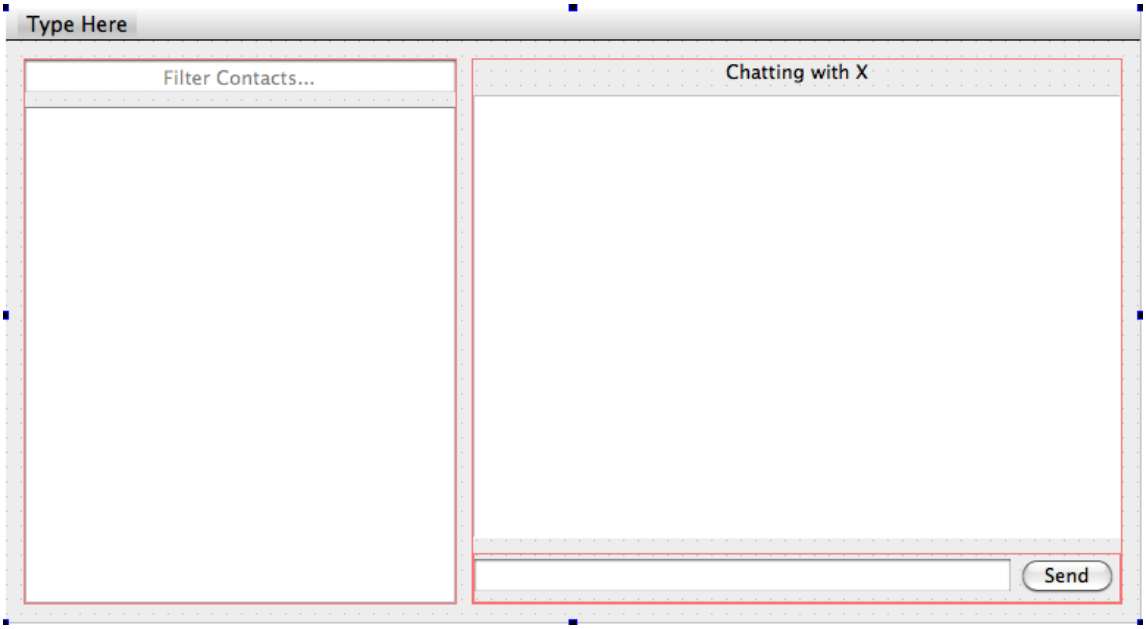

Figure 8 - The form for the main screen built in Qt Creator

Figure 8 shows the form laid out in Qt Creator. The red outlines designate layout groupings. On the left side, the space for the contact list is laid out, grouped with the text entry box for the contact filtering. Using the properties Qt provides for text entry boxes, the *Input Prompt* pattern is implemented to place the greyed out text in this box, which disappears upon the user typing in the field.

On the right side, the message space is laid out, with the message entry and send button beneath it. *Autocomplete* is implemented by default on the N900 since Qt uses the native APIs to generate the text entry box. The *Prominent "Done" Button* pattern is partially implemented here by providing the "Send" button. This button is then tied to the actual action of sending a message in code later on. *Titled Sections* are also provided here directly above the message area.

Also provided here is the *Diagonal Balance* pattern. This is achieved by structuring the layout so that the text entry areas are in opposite corners. Using the built in properties to

the form layouts, I divided the available space so that one third of the space was dedicated to the contact list area, and two thirds of the space was dedicated to the messaging area. The layout for the *Two Panel Selector* is implemented here, however the functionality for this will be implemented elsewhere using Qt's signals and slots to communicate the clicking of a contact in the list.

Once the form was completed, I wrote the code to make the interface elements functional. The conversation view uses Qt's built in QListView object to display the messages. Similarly, the contact list is implemented using Qt's QTreeView object. Both of these provide implementation for the *List-Based Layout* pattern. By using APIs for the Maemo OS, this also provides element sizing that is conducive to the *Finger Friendly Controls* pattern. The QTreeView also provides implementation for the *Closable Panels* pattern by providing the ability to set groups as parent items for their contacts, and have expandable and collapsible views of their contents. Internally, contacts, contact groups, and messages are all stored as independent classes, with references to each other where necessary.

The *Color Coded Sections* pattern is implemented in its own class using Qt's Delegate system. Qt structures interface elements using a Model-View-Delegate architecture. The view in this case is provided by the QTreeView and QListView classes, and is responsible for displaying the data in the layout space that it is placed within. The model is used to manage the data internally, so that more complex data can be used as the source for the view. A custom implementation of the default model is used to handle the data stored within the contact, contact group, and message objects. The delegate class is responsible for

both handling interaction with the view, as well as rendering individual view items. The default delegate is extended in order to override the painting functions, so that the custom *Color Coded Sections* can be implemented.

Due to time constraints, the other components of the prototype were not implemented. However, continuing with the current architecture, it is easy to see where they could be implemented. A conversation management class would be used to implement the dynamic "Active Chats" group, representing the *Smart Menu Items* pattern. A state management class could utilize the QSettings class provided by Qt to store data associated with application state (and perhaps might be used as the conversation management class), implementing the *Application State Management* pattern. Either in the same class, or in a similar class, the *Launch Process*, and in turn the *Inform the User About What is Happening* pattern, could be implemented, retrieving the previous application state upon application launch. Menus would be implemented by adding them in on the main screen form. On the N900 the menus default to being placed in a dropdown menu as described in *Finger Friendly Menu Choices*. When menu items require additional actions after being selected, those actions will be presented using *Modal Panels*.

Results

The goal of designing and prototyping an interface was to determine the effectiveness of designing with user interface design patterns. Given the time constraints of this project, there was not an opportunity to do any user testing in order to confirm that the final design and prototype were effective user interfaces. The interface has qualities that we can evaluate though. The majority of the actions a user will take when using the application are

presented on the main screen with simple one to two click access. Additional less frequent actions are presented within an easy to access menu system, and are no more than two levels deep into the interface. All of the interface elements are large enough to be easily readable, and finger-clickable. Additionally, the design is very similar to successful applications that serve the same purpose.

In order to test the efficiency of the application's interface, a Keyboard-Level Model (KLM) analysis was performed on two primary tasks: initiating a new conversation, and switching to another active conversation. The analysis was also performed on the AIM for iPhone instant messaging application, in order to provide a comparison with a current similar and popular application. The most significant difference between AIM for iPhone and the prototype application is that AIM for iPhone uses a software-based keyboard. However, keyboarding actions are not performed in this KLM analysis, so this should not affect the analysis. It is assumed that the starting point for each analysis is in an active conversation.

Initiate a New Conversation	
AIM for iPhone	**Prototype on Nokia N900**
Tap to return to Contact List	Scroll to Contact Group
Scroll to Contact Group	Tap to expand Contact Group
Tap to expand Contact Group	Tap Contact to initiate Conversation
Tap Contact to initiate Conversation	
Total = 4P = 4.40s	**Total = 3P = 3.30s**

Switch to Another Active Conversation	
AIM for iPhone	**Prototype on Nokia N900**
Tap to return to Contact List	Scroll to Active Chats
Scroll to Contact Group	Tap to expand Active Chats Group
Tap to open IMs tab	Tap Contact to open Conversation
Tap Contact to open Conversation	
Total = 4P = 4.40s	**Total = 3P = 3.30s**

Table 1 - KLM Analysis comparing AIM for iPhone and Prototype on Nokia N900

As is demonstrated by the KLM analysis, the prototype application is able to perform primary tasks in one fewer steps than AIM for iPhone. This is due to the fact that the contact list is always visible in the prototype, whereas AIM for iPhone dedicates the entire screen to either the contact list or the open conversation.

The design process itself was straightforward. The first step, as Barbara Ballard describes, is to understand how the hardware will limit or otherwise affect the interface. Then the applications primary goals were identified. From that point, with all the factors understood, the patterns that were potentially useful to the application design were selected from the pattern collection.

Using the selected patterns aided me in the design process in several respects. The patterns helped to reinforce good design practices, giving a more rigid set of boundaries to operate within. In a similar vein, the patterns reduced ambiguity in the design. Each element had a specific, well-defined purpose. In portions of the design where it was not as clear what the best approach to take was, patterns helped to provide a potential solution. As is demonstrated by the use of *Color Coded Section*, patterns can also help inspire a more creative design that introduces unique alternative features that may not have been introduced otherwise. The end result is a simple, clean design that is straightforward and easy to use.

Instant messaging applications are relatively common. Additionally, they are typically relatively unsophisticated interfaces, with very few extra controls or complex actions. These qualities were part of the reason that this type of application was chosen, so it would be easier to demonstrate a successful interface design in the absence of user testing.

However, these qualities also limit the extent to which design patterns can be demonstrated. This is because design patterns gain one of their largest advantages in providing solutions where solutions may not be obvious. With an interface that is as familiar as an instant messaging application, the problems that the interface presents have solutions that have also become familiar, reducing the need for using design patterns to solve them.

Another significant advantage of using design patterns that this project was not able to realize is that they are highly reusable. In the design phase of the project, the reuse is limited to the conceptual solution only. While this is very useful for solving common design problems, its utility in implementing the design is limited. In order to extend the utility of the patterns beyond the design phase, a reusable code base would need to be associated with the pattern. This code base would not need to be fully implemented, as it would be impractical to treat the patterns as cookie-cutter interface components in most cases. However, having some form of reusable code associated with the patterns would greatly increase their utility.

There were two major reasons why reusable implementations of the patterns could not be fully realized. The first reason is that the patterns used came in a variety of formats, and included a variety of different interpretations as to what a design pattern is. This decentralized nature to the patterns limits the ability to associate code with them. The patterns were also not presented with any particular programming language in mind, so there was little motivation to provide implementation examples with them. The second reason there was very little reuse is that the code for this project was written entirely from

scratch, with only this project in mind. In a professional development environment, where the patterns would be reused in similar manners, a code base could more easily be produced, and would provide greater utility.

While the patterns used did not specify any code that would aid in implementing them, some code reuse was demonstrated via the Qt Framework. Interface elements within Qt implemented many of the patterns automatically, such as the *List-Based Layout*. Implementing this pattern largely consisted of requesting the interface element from Qt, and writing the code necessary to adjust it to fit the specific needs of the application. Similarly, most of the interface element arrangements were handled automatically by Qt, with the ability to specify element groupings and relative sizes. Some patterns may require more customization than others, but having some level of reusable code to work from greatly reduces the required effort for implementation.

The most difficult portion of implementing the prototype was in deciding the best way to structure the code in order to implement the patterns that Qt did not provide directly, such as the *Color Coded Sections*. The initial screen was created using Qt Designer, arranging the desired elements onto a canvas and assigning them to layout groups. Once they were arranged in the canvas, layout properties were adjusted to produce the desired sizing and positional effects. Qt handled generating all the code for making these elements accessible to the rest of the program. Building the models to manage the contacts, contact groups, and messages within the program required the bulk of the work. However, this work was made more difficult due to my inexperience using Qt's Model classes, and an experienced Qt developer would have been able to produce the custom model much more

easily. Once the models were produced, extending the default Delegate class to implement the *Color Coded Sections* was relatively trivial, taking advantage of Qt's provided painting methods.

Conclusions and Future Work

Design patterns have been utilized with great success in architecture, as well as in software programming. Their extension into user interface design seems not only practical, but also natural. Since the late nineties, the user interface community has discussed the great potential for user interface design patterns. (Seffah) Yet, a decade later, design patterns have not made their way to the forefront of user interface design techniques as many expected. Instead, the community has tended to favor methodologies such as style guides, or relied simply on an understanding of good design principles.

The reason design patterns haven't taken off isn't because they aren't useful. It isn't even because the other techniques are better. One of the biggest problems design patterns face in the user interface community is that their development is so decentralized. When the Gang of Four released their seminal book "Design Patterns" for the software development community, it became the definitive model by which all other software design patterns were based. It was well written, and the patterns were unique and extremely useful. The user interface community has not experienced this same benefit. There have been a number of books and papers that furthered the discussion on design patterns, but each seems to take its own unique approach. This has resulted in a very loose definition of what a design pattern consists of, decreasing their utility to the community, and also

weakening the "brand" of design patterns, making them less enticing for designers to use and contribute to.

Even with this hindrance, design patterns have proven capable of being a successful approach to user interface design, as is evidenced in the Yahoo! case study. Here, the patterns gained strength by taking on a singular style for defining them. Each pattern adheres to the same structure, making it easier to familiarize oneself with new patterns due to familiarity with the format. The patterns are also described using similar language, in part due to the matching structure. The structure includes some categories that are specific to Yahoo! as well, an advantage of having the library be group-specific. The organization of the library presents another significant advantage, which doesn't exist for the general community. There have been a few different attempts at building user interface design pattern libraries, but none of them seem to have resulted in common use.

One area that user interface design patterns are lacking in is the lack of complex patterns. The best patterns help developers to solve difficult problems, because chances are if the problem were easy to solve, the developer wouldn't need help solving it. Most of the patterns that I encountered were useful, but described simple solutions to simple problems, such as the *List-Based Layout* pattern. They definitely have a role as design patterns, and the more strict definition that comes along with making something a design pattern helps to increase the understanding of these solutions, and the circumstances under which they are useful. However, this advantage is not significant enough to make a design pattern collection consisting of just this type of pattern worthwhile. The collection

would have to include patterns that utilize the simpler patterns as components, or otherwise complement them.

Work on design patterns in user interface design is still very active in the development community today, as evidenced by the PEICS conference, which started in 2010. Their "Topics of Interest" section states that they are interested in:

- Development of HCI patterns
- Languages for the definition of HCI patterns
- Pattern representation: UML, XML, USiXML, mathematical formalization etc.
- Tools supporting the development of HCI patterns
- Pattern-oriented design and engineering
- The combination of models and HCI patterns in a model-driven development process
- Patterns in practices (project experience): Web services, mobile applications, etc.

Of these topics, the most important to consider are the development of HCI patterns, and the languages for the definition of HCI patterns. As the Yahoo! case study shows, having a unified language for defining patterns is a key aspect for making it useful. The development of more user interface patterns, especially those patterns that solve difficult problems, is equally important, since this is one of the biggest strengths of patterns in general. In my research, the use of design patterns in user interface design very much has the feel of a leak in a dam that has been trickling for the past decade. There has been a slow progression forward as the idea percolates through the community, gaining more interest as it goes along. So far, patterns have had limited impact, but there is a huge amount of potential behind the idea, just waiting for the final push through that allows that potential to be brought to fruition.

Appendices

Appendix A – Describing Design Patterns

Pattern Name and Classification

The pattern's name conveys the essence of the pattern succinctly. A good name is vital, because it will become part of your design vocabulary.

Intent

A short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?

Also Known As

Other well-known names for the pattern, if any.

Motivation

A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem. The scenario will help you understand the more abstract description of the pattern that follows.

Applicability

What are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations?

Structure

A graphical representation of the classes in the pattern using a notation based on the Object Modeling Technique (OMT). We also use interaction diagrams to illustrate sequences of requests and collaborations between objects.

Participants

The classes and/or objects participating in the design pattern and their responsibilities.

Collaborations

How the participants collaborate to carry out their responsibilities

Consequences

How does the pattern support its objectives? What are the trade-offs and results of using the pattern? What aspect of system structure does it let you vary independently?

Implementation

What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?

Sample code

Code fragments that illustrate how you might implement the pattern in C++ or Smalltalk

Known Uses

Examples of the pattern found in real systems. We include at least two examples from different domains.

Related Patterns

What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

Appendix B – Mobile Heuristics

Visibility of system status and losability/findability of the mobile device

Through the mobile device, the system should always keep users informed about what is going on. Moreover, the system should prioritize messages regarding critical and contextual information such as battery status, network status, environmental conditions, etc. Since mobile devices often get lost, adequate measures such as encryption of the data should be taken to minimize loss. If the device is misplaced, the device, system or application should make it easy to find it back.

Match between system and the real world

Enable the mobile user to interpret correctly the information provided, by making it appear in a natural and logical order; whenever possible, the system should have the capability to sense its environment and adapt the presentation of information accordingly.

Consistency and mapping

The user's conceptual model of the possible function/interaction with the mobile device or system should be consistent with the context. It is especially crucial that there be a consistent mapping between user actions/interactions (on the device buttons and controls) and the corresponding real tasks (e.g. navigation in the real world).

Good ergonomics and minimalist design

Mobile devices should be easy and comfortable to hold/ carry along as well as robust to damage (from environmental agents). Also, since screen real estate is a scarce resource, use it with parsimony. Dialogues should not contain information which is irrelevant or rarely needed.

Ease of input, screen readability and glancability

Mobile systems should provide easy ways to input data, possibly reducing or avoiding the need for the user to use both hands. Screen content should be easy to read and navigate through notwithstanding different light conditions. Ideally, the mobile user should be able to quickly get the crucial information from the system by glancing at it.

Flexibility, efficiency of use and personalization

Allow mobile users to tailor/personalize frequent actions, as well as to dynamically configure the system according to contextual needs. Whenever possible, the system should support and suggest system based customization if such would be crucial or beneficial.

Aesthetic, privacy and social conventions

Take aesthetic and emotional aspects of the mobile device and system use into account. Make sure that user's data are kept private and safe. Mobile interaction with the system should be comfortable and respectful of social conventions.

Realistic error management

Shield mobile users from errors. When an error occurs, help users to recognize, to diagnose, if possible to recover from the error. Mobile computing error messages should be plain and precise. Constructively suggest a solution (which could also include hints, appropriate FAQs, etc). If there is no solution to the error or if the error would have negligible effect, enable the user to gracefully cope with the error.

Bibliography

Wania, Christine E., and Michael E. Atwood. "Pattern Languages in the Wild: Exploring Pattern Languages in the Laboratory and in the Real World." *DESRIST.* Malvern: ACM, 2009.

Ballard, Barbara. "Mobile User Interface Design Patterns." In *Designing the Mobile User Experience*, 95-131. Chichester: Wiley, 2007.

Bertini, Enrico, Silvia Gabrielli, and Stephen Kimani. "Appropriating and Assessing Heuristics for Mobile Computing." *AVI '06.* Venezia: ACM, 2006. 119-126.

Borchers, Jan O. "A Pattern Approach to Interaction Design." *AI & Society*, 2001: 359-376.

Breiner, Kai, Marc Seissler, Gerrit Meixner, Peter Forbrig, Ahmed Seffah, and Kerstin Klöckner. "PEICS Towards HCI Patterns into Engineering of Interactive Systems." *PEICS '10.* Berlin: ACM, 2010. 1-3.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Second Printing. Reading, MA: Addison-Wesley Publishing Company, 1995.

Kane, Shaun K., Jacob O. Wobbrock, and Ian E. Smith. "Getting Off the Treadmill: Evaluating Walking User Interfaces for Mobile Devices in Public Spaces." *MobileHCI.* Amsterdam: ACM, 2008. 109-118.

Leacock, Matt, Erin Malone, and Chanel Wheeler. "Implementing a Pattern LIbrary in the Real World: A Yahoo! Case Study." *ASIS&T IA Summit.* Montreal, 2005.

Nilsson, Erik G. "Design Patterns for User Interface for Mobile Applications." *Advances in Engineering Software* 40, no. 12 (2009): 1318-1328.

Seffah, Ahmed. "The Evolution of Design Patterns in HCI: Pattern Languages to Pattern-Oriented Design." *PEICS '10.* Berlin: ACM, 2010. 4-9.

Tidwell, Jenifer. *Designing Interfaces.* 2nd Edition. Sebastopol, CA: O'Reilly Media, Inc, 2006.